Athens University of Economics and Business Dept. of Informatics
M.Sc. Program in Data Science
INF322: Social Network Analysis Fall 2019
Instructor: Katia Papakonstantinopoulou

Homework 2: Apache Giraph

An iterative graph processing system built for high scalability

Due 11:59pm EEST December 9, 2019

General Instructions

Your answers should be as concise as possible.

Submission instructions: You should submit your report and code in a compressed directory uploaded via `https://eclass.aueb.gr`

Submitting answers: Prepare a report on your homework in a single PDF file named *hw2.pdf*

Submitting code: Prepare a `.txt` file named according to the instructions with the answer to the first problem and the completed Java files `SimpleBoggleComputation.java` and `GiraphAppRunner.java` for the second problem.

Problems

In this homework you will be using and developing algorithms with Apache Giraph, the open-source counterpart of Google's Pregel, that is mainly contributed by Facebook. Apache Giraph's source code, Apache Hadoop and everything you will need for this homework is preinstalled in the Virtual Machine provided by this course.[1] An Eclipse Java Project configured to execute Giraph jobs locally is also offered to you to allow for faster development.[2] Alternatively, you can proceed with installing Apache Giraph to your own platform by following the guidelines of this link: `http://giraph.apache.org/quick_start.html`.

[1] `https://goo.gl/CGm1YR`
[2] `https://drive.google.com/open?id=0B9VaiGKGwsiwNTQ3MDd1QTVKWUE`

1 Use Apache Giraph SimpleShortestPaths algorithm

The first part of this homework requires that you familiarize yourselves with executing Apache Giraph *jobs*. The instructions will guide you through the entire procedure and you will have to submit the output of the execution.

First you need to start the *Hadoop Filesystem* and *Map Reduce*. To this end you will need to use the respective *scripts* that are placed in the hadoop installation directory (/usr/local/hadoop/).

CAREFUL: Hadoop related tasks must be executed through a dedicated hadoop user (hduser in the VM). To switch users you have to issue the `su` command:

```
$ su hduser
```

The scripts you need to run are under the `bin/` directory and are:

1. `start-dfs.sh`
2. `start-mapred.sh`

CAREFUL: The scripts must be initiated in this order and terminated in the reverse.

Then, you can verify that everything is up and running by issuing the `jps` command. You should see the following processes running:

- NameNode
- JobTracker
- DataNode
- SecondaryNameNode
- Jps
- TaskTracker

Now, you can use the web browser to see a web interface of both the filesystem and the map reduce administration. The local addresses are already bookmarked in the VM's browser but are also listed below:

- Filesystem: `http://localhost:50070`
- Map Reduce Administration: `http://localhost:50030`

If you navigate in the filesystem after following the first link, you will find that a sample graph is already present there (tiny_graph.txt). This is the graph that you have to use for the first part of this homework.

In particular, you will execute the `SimpleShortestPaths` algorithm that is already implemented in Apache Giraph. To do that, run (as hduser) the following command:

```
/usr/local/hadoop/bin/hadoop jar /home/sna/eclipse/workspace/giraph/giraph-examples/target/giraph-examples-1.1.0-for-hadoop-1.2.1-
jar-with-dependencies.jar org.apache.giraph.GiraphRunner org.apache.giraph.examples.SimpleShortestPathsComputation
-vif org.apache.giraph.io.formats.JsonLongDoubleFloatDoubleVertexInputFormat -vip /user/hduser/input/tiny_graph.txt
-vof org.apache.giraph.io.formats.IdWithValueTextOutputFormat -op /user/hduser/output/shortestpaths -w 1
```

This command provides Hadoop with a jar file and specifies the class of the jar file that is to be executed (`org.apache.giraph.GiraphRunner`). This class is provided with another Giraph class
(`org.apache.giraph.examples.SimpleShortestPathsComputation`)
that extends: `org.apache.giraph.graph.BasicComputation`.
Then, some arguments specify the vertex input format (-vif), the vertex input path (-vip), the vertex output format (-vof), the output path (-op) and the number of workers (-w). As this is a pseudo-distributed hadoop installation with only one worker, you cannot increase the number of workers to more than one.

After you have executed this command, and if everything is done according to the instructions, there will be a file in your HDFS in the path:
`"/user/hduser/output/shortestpaths"`, with the result of your computation. For this part of the homework you have to submit this file. Browse through the hadoop filesystem and retrieve the file using your web browser. Alternatively, you can copy the file to your local filesystem using `hadoop dfs` command.

In addition, you should rename the file you retrieved to match the task id that saved your results. To retrieve the task id, browse through the job tracker to find your completed job, click on its id, then click on the completed map tasks, and there you will see the id of the task that saved your vertices. **Use it to name your file**. This will familiarize you with browsing through the details of a job's tasks where you can access useful information regarding the tasks you initiated, such as logs and errors.

2 Implement SimpleBoggleComputation

Boggle is a popular word game in which any number of players attempt to find words in sequences of adjacent letters in a grid of lettered dice such as the one shown in Figure 1.

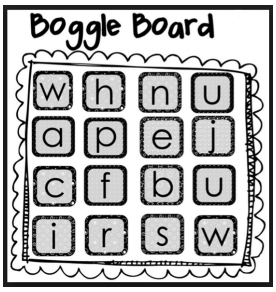

Figure 1: A Boggle board.

Note that a player can move to any of (maximum) 8 adjacent letters, but a word should not have multiple instances of same cell. For example, a player can go from the letter 'i' in the lower left corner of the grid to any of letters 'c', ''f', and 'r'. However, if the player has already used one of these letters earlier for the current word, the respective move would not be possible.

For the second part of this homework, you will have to implement an iterative Pregel-like algorithm that finds all words contained in a dictionary that are also present in the Boggle board. Consider that each cell in the board is a graph vertex and its neighbors are all the adjacent letters of the board.

You have to create a Java class that extends:
`org.apache.giraph.graph.BasicComputation`.
Such a class is provided with this homework and already takes care of certain implementation details. You should complete the implementation of this project with the following functionality:

- Every vertex should send an initial message to all its neighbors with a word starting with its respective letter, and the vertex id (so that the same vertex is not used again in this word).
- For every message a vertex receives, it should add its letter to the word of the message and examine if the word is in the dictionary. If so, it should add the word to the value of the vertex.[3] If the word is a prefix of a word in the dictionary then an updated message should be sent to all vertices that have not received it previously.

[3]Note that the type value of the vertex is defined as a list of Text objects.

Your algorithm should be able to find words for any input graph and any dictionary. However, a sample graph (boggle.txt) and an extremely small dictionary is provided with this homework, to enable discussion of results on a common ground and ease development.

Hints:

- As multiple cells may represent the same letter, the input graph uses the board coordinates to differentiate between cells. Extracting the letter is just a matter of getting the first character of the vertex id.
- The provided Java file specifies the data types that allow you to implement this algorithm. Pay special attention to the use of the `TextArrayListWritable` that enables you to send multiple `Text` objects (a word or vertex ids) using only one message. The provided Java file populates such a structure and retrieves its elements as an example for your implementation.
- Think of the number of messages that need to be created and the information they should contain. Initially, vertices need to send to all their neighbors a word starting from their letter. Then, each word can be appended by every vertex that receives it. However, words should not use each vertex more than once, so the messages should include the ids of the previously visited vertices.
- The input type of this algorithm is different than that in the first part of the homework. You can use the one that is provided to you with this homework instead.
- The dictionary is defined as a TreeSet. Look up methods *contains()* and *subSet()* as you may find them useful.

There are two alternatives you can follow for the second part of this homework:

If you plan to submit your algorithm for execution to hadoop, you should copy all Java files to their respective giraph directory and build giraph. For instance, `SimpleBoggleComputation.java` should go to:

/home/sna/eclipse/workspace/giraph/giraph-examples/src/main/java/org/apache/giraph/examples/

To build Apache Giraph after performing changes to the source code you have to execute the following from a terminal in the directory of giraph (/home/sna/eclipse/workspace/giraph) as the `sna` user:

```
mvn package -DskipTests
```

For your convenience an Eclipse Java Project is available here:
`https://drive.google.com/open?id=0B9VaiGKGwsiwNTQ3MDd1QTVKWUE`.

This project allows for local execution of giraph jobs enabling you to speed-up development. To use it, you should copy the the project into the '/home/sna/eclipse/workspace' directory and then import it into Eclipse (File ->Import ->Existing Projects Into Workspace). Then you can execute giraph jobs by running GiraphAppRunner.java as a Java Application. This class is configured to execute the example of the first part of this homework. You should make some minor adjustments to execute the code of the second part.

Bonus Question: Create a graph for a 4×4 boggle board of your choice and use a larger dictionary, e.g., `https://raw.githubusercontent.com/hillmanov/go-boggle/master/dictionary.txt` to find words in it. Submit your graph file, the board that it stands for, and the result of your execution.